The importance of reproducibility in computational science is being more and more recognized, which I think is a good sign. However, I also notice a lot of confusion about what reproducibility means exactly, and also confusion about the difference (if any) between reproducibility and replicability. I don’t see a consensus yet about the exact meaning of these terms, but I would like to give my own definitions and justify them by putting them into the general context of computational science.

I’ll start with the concept of reproducibility as it was used in science long before computers even existed. It refers to the reproducibility of the conclusions of a scientific study. These conclusions can take very different forms depending on the question that was being explored. It can be a simple “yes” or “no”, e.g. in answering questions such as “Is the gravitational force acting in this stone the same everywhere on the Earth’s surface?” or “Does ligand A bind more strongly to protein X than ligand B?” It can also be a number, as in “What is the lattice energy of NaCl?”, or a mathematical function, as in “How does a spring’s restoring force vary with elongation?” Any such result should come with an estimation of its precision, such as an error bar on numbers, or a reliability estimate for a yes/no answer. Reproducing a scientific conclusion means finding a “close enough” answer by performing “similar” experiments and analyses. As the terms “close enough” and “similar” show, reproducibility involves human judgement, which may well evolve over time. Reproducibility is thus not an absolute feature of a specific result, but the evaluation of a result in the context of the current state of knowledge and technology in a scientific domain. Every attempt to reproduce a given result independently (different people, tools, methods, ...) augments scientific knowledge: If the reproduction leads to a “close enough” results, it provides information about the precision with which the results can be obtained, and if if doesn’t, it points to some previously unrecognized crucial difference between the two experiments, which can then be explored.

Replication refers to something much more specific: repeating the exact steps in an experiment using the same (or equivalent) equipment, and comparing the outcomes. Replication is part of testing an experimental setup, or a form of quality assurance. If I measure the same quantity ten times using the same equipment and experimental samples, and get ten slightly different values, then I can use these numbers to estimate the precision of my equipment. If that precision is not sufficient for the purposes of my planned scientific study, then the equipment is not suitable.

It is useful to describe the process of doing research by a two-layer model. The fundamental layer is the technology layer: equipment and procedures that are well understood and whose precision is known from many replication attempts. On top of this, there is the research layer: the well-understood equipment is used in order to obtain new scientific information and draw conclusions from them. Any
scientific project aims at improving one or the other layer, but not both at the same time. When you want to get new scientific knowledge, you use trusted equipment and procedures. When you want to improve the equipment or the procedures, you do so by doing test measurements on well-known systems. Reproducibility is a concept of the research layer, replicability belongs to the technology layer.

All this carries over identically to computational science, in principle. There is the technology layer, consisting of computers and the software that runs on them, and the research layer, which uses this technology to explore theoretical models or to interpret experimental data. Replicability belongs to the technology level. It increases trust in a computation and thus its components (hardware, software, overall workflow, provenance tracking, ...). If a computation cannot be replicated, then this points to some kind of problem:

1. different input data that was not recorded in the workflow (interactive user input, a random number stream initialized from the current time, ...)
2. a bug in the software (uninitialized variables, compiler bugs, ...)
3. a fault in the hardware (an unreliable memory chip, a design flaw in the processor, ...)
4. an ambiguous specification of the result of the computation

Ideally, the non-replicability should be eliminated, but at the very least its cause should be understood. This turns out to be very difficult in practice, in today's computing environments, essentially because case 4 is frequent and hard to avoid (today's popular programming languages are ambiguous), and because case 4 makes it impossible to identify cases 2 and 3 with certainty. I see this as a symptom of the immaturity of today's computing environments, which the computational science community should aim to improve on. The technology for removing case 4 exists. The keyword is "formal methods", and there are first attempts to apply them to scientific computing, but this remains an exotic approach for now.

As in experimental science, reproducibility belongs to the research layer and cannot be guaranteed or verified by any technology. In fact, the "reproducible research" movement is really about replicability - which is perhaps one reason for the above-mentioned confusion.

There is at the moment significant disagreement about the importance of replicability. At one end of the spectrum, there is for example Ian Gent's recomputation manifesto, which stresses the importance of replicability (which in the context of computational science he calls recomputability) because building on past work is possible only if it can be replicated as a first step. At the other end, Chris Drummond argues that replicability is "not worth having" because it doesn't contribute much to the real goal, which is reproducibility. It is worth reading both of these papers, because they both do a very good job at explaining their arguments. There is actually no contradiction between the two lines of arguments, the different conclusions are due to different criteria being applied: Chris Drummond sees replicability as valuable only if it improves reproducibility (which indeed it doesn't), whereas Ian Gent sees value in it for a completely different reason: it makes future research more efficient. Neither one mentions the main point in favor of replicability that I have made above: that replicability is a form of quality assurance and thus increases trust in published results.

It is probably a coincidence that both of the papers cited above use the term "computational experiment", which I think should best be avoided in this context. In the natural sciences, the term "experiment" traditionally refers to constructing a setup to observe nature, which makes experiments the ultimate source of truth in science. Computations do not have this status at all: they are applications of theoretical models, which are always imperfect. In fact, there is an interesting duality between the two: experiments are imperfect observations of the ultimate truth, whereas computations are, in the absence of buggy or ambiguous software, perfect observations of the consequences of imperfect models. Using the same term for these two concepts is a source of confusion, as I have pointed out earlier.

This fundamental difference between experiments and computations also means that replicability has a
different status in experimental and computational science. When doing imperfect observations of nature, evaluating replicability is one aspect of evaluating the imperfection of the observation. Perfect observation is impossible, both due to technological limitations and for fundamental reasons (any observation modifies what is being observed). On the other hand, when computing the consequences of imperfect models, replicability does not measure the imperfections of the model, but the imperfections of the computation, which can theoretically be eliminated.

The main source of imperfections in computations is the complexity of computer software (considering the whole software stack, from the operating system to the scientific software). At this time, it is not clear if we will ever succeed in taming this complexity. Our current digital computers are chaotic systems, in which even the tiniest change (flipping a bit in memory, or replacing a single character in a program source code file) can change the result of a computation beyond any bounds. Chaotic behavior is clearly an undesirable feature in any scientific equipment (I can't think of any experimental apparatus suffering from it), but for computation we currently have no other choice. This makes quality assurance techniques, including replicability but also more standard software engineering practices such as unit testing, all the more important if we want computational results to be trustworthy.

Explore posts in the same categories: Computational science, Reproducible research, Science